A Generic Platform for Scalable Access to Multimedia-on-Demand Systems

Raouf Boutaba and Abdelhakim Hafid, *Member, IEEE*

***Abstract*— Access to multimedia servers is commonly done according to a client/server model where the end user at the client host retrieves multimedia objects from a multimedia server. In a distributed environment, a number of end users may need to access to a number of multimedia servers through one or several communication networks. Such a scenario reveals the requirement for a distributed access platform. In addition, the demand for multimedia information is increasing beyond the capabilities of high performance storage devices. Therefore, load distribution and scalability issues must be addressed while designing and implementing the distributed access platform. This paper introduces a scalable access platform (SAP) for managing user access to multimedia-on-demand systems while optimizing resource utilization. The platform is generic and capable of integrating heterogeneous multimedia servers. SAP operation combines static replication and dynamic load distribution policies. It provides run time redirecting of client requests to multimedia servers according to the workload information dynamically collected in the system. To support multimedia-on-demand systems with differing quality-of-service (QoS) requirements, the platform also takes into account, as part of the access process, user QoS requirements and cost constraints. This paper also presents an application of the generic platform implementing a scalable movie-on-demand system, called SMoD. Performance evaluation based on simulation shows that in many cases SMoD can reduce the blocking probability of user requests, and thus can support more users than classical video-on-demand (VoD) systems. It also shows that the load is better distributed across the video servers of the system.**

***Index Terms*—Load distribution, multimedia-on-demand, quality-of-service (QoS), scalability, service access.**

I. Introduction

RECENT developments in computer and communication technologies have rendered possible the provision of multimedia-on-demand services. Typically, a multimedia server reserves a certain amount of resources to deliver a multimedia stream with certain qulaity-of-service (QoS) guarantees. The above implies that a multimedia server can support only a limited number of users depending on its capacity, e.g., CPU, bandwidth, etc. A large number of studies have concentrated on the performance of multimedia servers. Most of the proposed approaches use costly storage devices, supercomputers, and massively parallel memory and I/O systems.

Manuscript received May 1, 1998; revised April 1, 1999.

R. Boutaba is with the Electrical and Computer Engineering Department, University of Toronto, Toronto, Ont., M5S 3G4 Canada (e-mail: rboutaba@comm.utoronto.ca).

A. Hafid is with the Electrical and Computer Engineering Department, University of Western Ontario, London, Ont., N6A 5B9 Canada.

Publisher Item Identifier S 0733-8716(99)05599-7.

However, the demand for multimedia applications is increasing even beyond the capabilities of high performance multimedia servers. Therefore, the trend is toward replicating multimedia servers. Future multimedia-on-demand environments are envisioned to be composed of a large number of client hosts and a large number of heterogeneous multimedia server instances connected through high-speed networks. The majority of emerging multimedia applications, despite their different logic, would be based on a client/server communication model. This paradigm requires a number of common multimedia support services ranging from high-speed transport to real-time storage and processing services. Today's challenge is particularly the provision of the system software necessary to support and facilitate the creation, evolution, and management of multimedia services and applications. In essence, our work falls into this category. More specifically, a system approach is proposed to address the management of user access to distributed multimedia-on-demand systems while optimizing resources utilization.

The shift toward distributed client/server architectures to implement multimedia-on-demand systems stresses the requirement for distributed access platforms to hide the individuality and heterogeneity of media servers to end users. Such access platform has to be scalable so as to support the ever increasing number of users and their varying requirements. In this paper, we introduce a scalable access platform (SAP). SAP is a generic platform for scalable access to multimedia-on-demand systems, designed to overcome the limitations of current multimedia servers. It builds on existing heterogeneous media server technologies to provide a uniform multimedia on-demand system to end users. In addition, adding a new server technology to SAP only requires a mapping between SAP's generic protocol and the server's proprietary access protocol.

The scalability of SAP is addressed through a static replication of multimedia objects and a dynamic load distribution across the multimedia servers in the system. Load distribution is particularly important due to the nonuniform distribution of users' requests, which leads to load imbalances among servers and poor utilization of the overall system resources. Typically, highly loaded servers might reject some service requests when lightly loaded servers are available. This leads to inefficient utilization of the available server resources and a higher blocking probability of users' requests.

It is the objective of SAP to support scalable access to multimedia objects while distributing the load across the multimedia servers available in the system. SAP assumes that the most frequently requested multimedia objects are

replicated on a large number of servers. In addition, for each user request, SAP supports the dynamic selection of the most appropriate multimedia server to handle this request. Several criteria are considered in the selection process including: the user's QoS requirements, the user's host capabilities to display the requested multimedia object, the cost constraints, and the current workload of the multimedia servers. The result of the selection process is the least loaded multimedia server in the system, which satisfies the previous criteria. A user request is rejected only if none of the servers in the system can satisfy the request, which reduces the overall blocking probability of service requests.

In order to reduce the overhead introduced by the load distribution capability, i.e., the exchange of load information, multimedia servers are aggregated into management domains. This also reduces the complexity of the overall access management task. Each domain is responsible for managing access to the multimedia servers it aggregates, as well as for distributing the load among them. The various domains are then brought together into a hierarchy of domains to provide a system-wide access. To allow for such organization of user's access, SAP introduces intermediate access agents and managers between multimedia clients and servers. The managers are hierarchically distributed in the system for directing user requests to the most appropriate multimedia servers. Based on the system state information exchanged in the frame of access management and load distribution procedures, SAP is also capable of recovering automatically from QoS degradations and multimedia server failure.

The SAP platform is also generic in that it can be applied to support a variety of multimedia applications such as video-on-demand (VoD) or news-on-demand. A demonstrator application of the SAP capabilities has been implemented and deployed to provide a scalable movie-on-demand system (SMoD). Simulation-based performance measures show the efficiency of SMoD compared to classical VoD systems. The blocking probability of user requests is minimized, and the overall SMoD availability is increased.

The paper is organized as follows. Section II discusses related work, particularly emphasizing load distribution and scalability issues. Section III addresses the building of a middleware between media-on-demand clients and servers. It describes the SAP platform designed for efficient load-balancing and resource discovery. Section IV presents the load distribution policies as defined for the access platform. Section V describes the implementation and deployment of a scalable SMoD using SAP. In Section V, we also evaluate the performance of SAP/SMoD through simulations. Finally, Section VI concludes the paper and gives directions for future research.

II. Related Work

The demand for multimedia information is increasing beyond the capabilities of high performance storage devices. It leads to the necessity of replicating server instances, using high-speed networks. This replication is intended to present a uniform storage architecture to the client, by hiding the distribution of servers [1]. In this perspective, load distribution and scalability issues have to be addressed. This section overviews classical approaches for distributing the load across servers and presents some of the ongoing work addressing scalability issues in multimedia-on-demand environments.

Load distribution algorithms can be classified into static and dynamic. In static algorithms, task assignment decisions are made *a priori* and are not changed during run time. In contrast, dynamic algorithms use current system workload information for run time assignment of tasks to appropriate servers. Therefore, despite the higher run-time complexity, dynamic algorithms can provide better performance than static algorithms. Several existing resource optimization approaches implement a dynamic load distribution algorithm [2]–[4]. One crucial problem in distributed environments is the unavailability of an accurate and timely global state information. Therefore, the majority of load distribution approaches is heuristic based, and hence, it provides suboptimal performance only. Taxonomy of dynamic suboptimal heuristic-based load distribution algorithms is given in [5].

Three issues have to be considered for the design of a dynamic load distribution facility:

1) the load information collection policy, which refers to the model adopted for the representation of the workload information and the frequency at which the workload state information is locally updated;
2) the load information exchange policy, which refers to the way workload state information is disseminated among the servers in the system;
3) the task placement policy, which refers to the task assignment decision based on the workload information.

The workload of a server is commonly modeled as the number of tasks currently supported by this server [6]. This includes both executing tasks as well as those waiting for execution in the server's queue. The server's processing speed is also an important parameter for the evaluation of the workload. Indeed, the time necessary for the execution of the same task by different servers depends on the performance of these servers. Therefore, the workload of a server is more accurately computed as the number of processes currently executed divided by the speed of the server [4]. However, this load information model does not take into account the size of the tasks in terms of processing time. Indeed, a server executing a single processing power demanding task could be more loaded than another server executing a number of lightweight tasks. As we will show in the Section IV, general QoS requirements associated with service requests can be used to determine a more accurate workload information.

Most of existing policies for information exchange are polling based. Polling can be initiated by a lightly loaded server to locate a heavily loaded one, or it can be initiated by a heavily loaded server to locate an idle one [4], [7]. The first approach is called server-initiated polling, while the second one is called client-initiated polling. Eager *et al.* [7] demonstrated that none of these approaches performs consistently over the whole range of system workload. A combination of these two basic approaches is proposed in [2] where client-initiated polling only occurs at low system workload, whereas

server-initiated polling is performed whenever appropriate. Most polling-based load distribution policies use a polling limit, usually defined as a function of the number of servers in the system [2], to control the number of pollings. Indeed, systems with a large number of servers involve a large number of pollings leading to a high overhead in terms of network bandwidth and CPU, which may cancel the performance gain obtained via load distribution.

The previous considerations reveal a major deficiency of polling-based load distribution algorithms, i.e., the lack of scalability. This paper aims to show how scalability issues can be addressed using the "divide and conquer" philosophy. In our approach, the domain concept is introduced as a flexible means for grouping servers according to geographical, organizational, or servers' performance criteria. Other criteria for grouping servers into domains and their tradeoffs are discussed in [8]. The load distribution is based on a server-initiated polling mechanism implemented for each individual domain. The various domains are then organized into a hierarchy of domains to provide a system-wide load distribution facility. The size of the domains can be determined and/or changed to modulate the overhead introduced by the polling compared to the performance of the polling-based load distribution. The domain hierarchy may contain as many domain levels as necessary according to the size and nature of the overall system, e.g., the number of servers, their distribution, the network topology, etc.

The majority of task placement policies are threshold-based, i.e., a server is considered a candidate for receiving and processing new tasks if its workload is below the threshold [2], [9]. However, using a single fixed threshold is not appropriate in case of rapid load fluctuations. Indeed, rapid fluctuations, up and down around the threshold, would base the decision-making and induce frequent oscillations. Therefore, double thresholds have been introduced to provide a tolerance of state fluctuation [10].

In this paper, load distribution is limited to tasks corresponding to service requests issued by clients to retrieve and display multimedia objects. However, the dynamic load distribution components discussed earlier are relevant because our approach combines both replication strategies with run time redirecting of client requests. The latter are submitted to multimedia servers according to the workload information dynamically collected in the system. Server and file replication are the common mechanisms used to provide scalable multimedia storage.

Multimedia storage architectures have recently been the subject of numerous studies, particularly addressing the real-time demands of audio and video. Several high performance multimedia storage systems have been designed to increase the bandwidth and storage capacity of single disks [11]–[14]. However, the scalability of such systems is limited since disks cannot be incrementally added to provide higher bandwidth and storage facilities [1]. Undergoing work addresses the scalability issue of multimedia storage by replicating multimedia servers in the system and by providing a uniform access interface to end users. These approaches are similar to file replication and placement techniques used in conventional file servers, such as the static replication of all files on all servers in the system [15] or the replication of the frequently accessed files on all servers [16]. However, the latter techniques have been designed to retrieve classical data files, and are not adapted for real-time access and retrieval of multimedia objects, e.g., audio and video.

Few recent approaches have to be mentioned here as they have particularly addressed the scalability issue for multimedia storage architectures [17]–[19]. The work presented in this paper falls into this category. Dan and Sitaram [17] defined a static scheme for the placement of video objects based on statistical information. This scheme is complementary to our proposal, which provides a similar scheme for the replication of multimedia objects. However, this scheme reduces load imbalances but cannot eliminate them, since it implements a static statistics-based replication algorithm. Therefore, our approach adds a dynamic load distribution facility to the static replication of frequently accessed multimedia objects.

In the Berkeley approach [18], storage management algorithms have been defined for hierarchical distributed VoD systems. The algorithms manage the distributed cache in video servers and provide a video placement capability, which selects a video server on which to place a requested video according to the servers' load, network load, and service-wait time. The Berkeley VoD system is targeted for enterprise VoD environments, and hence it emphasizes the provision of access to a large number of videos by a small number of users. In contrast, emphasis in our approach is on the rapid access to a small number of multimedia objects by a large number of users. A typical scenario in which our approach is suitable is the provision of access to SMoD servers. In general, our approach is suitable to provide access to commercial media-on-demand servers targeting large numbers of users.

In the latter perspective, a scalable hierarchical video storage architecture has been defined at Lancaster University [13], which is based on a three-level distribution hierarchy supporting both file replication and network node striping for balancing the load across multimedia server instances. Network striping implements the redundant arrays of inexpensive disks (RAID) concept [20], but operates at the level of the network. The striping technique consists to split individual multimedia files into pieces and to distribute them across server instances, therefore allowing the sharing of a stream across these instances. Our approach is different in that it is targeting continuous multimedia file servers and abstracts from the storage details of multimedia objects.

Finally, commercial video servers are also available, for example, from Microsoft [19], Oracle [21], and Silicon Graphics [22], etc. These products basically support traditional file system operations, but also support real-time file access as well as the provision of data streams at a guaranteed rate. While the Tiger Video file server from Microsoft is built entirely of hardware components, Oracle and Silicon Graphics use supercomputer and massively parallel memory and I/O systems. Both approaches are not scalable. In contrast, our approach promotes the use of hierarchically organized multimedia servers distributed across the network. By means of a load distribution facility, optimized access to server

instances is provided by a software platform, which acts as an intermediate service between multimedia servers and clients. The previous products and others can be interfaced by our SAP.

In general, our platform can be distinguished by the objectives pursued for its design. Indeed, SAP is intended to be generic, capable of integrating heterogeneous multimedia servers by providing appropriate gateways, and scalable in managing user access to multimedia-on-demand systems, while optimizing resource utilization. A key aspect in the design of SAP is to take into account user QoS requirements in the access process, the objective being to support multimedia-on-demand systems with differing QoS requirements. Among the QoS constraints, only those related to the end systems (client and server systems) capabilities are considered within SAP. The platform builds on existing operating systems and existing continuous media file systems to provide scalable access with a better performance.

In this paper, the transport network capacity and access requirements in terms of bandwidth are not considered. SAP assumes that there are no delivery bandwidth constraints on the communication network, e.g., between client hosts and media servers. Although this is a rather simplistic assumption, it allows us to focus on the system level of the platform. Some research groups have addressed the bandwidth constrained multimedia-on-demand systems including [10], [18], [23]. Another parameter used as part of SAP procedure is the cost of user access to multimedia objects, which should include the costs of retrieval and delivery. However, this paper does not address pricing considerations; it only considers the cost as a parameter during the negotiation of user access to a multimedia object. An investigation of costs of storing and transporting objects in distributed multimedia-on-demand systems is done in [23].

III. SAP Distributed Architecture

Access to multimedia servers is commonly done according to a client/server model where the end user at the client host retrieves multimedia objects from a multimedia server. In a distributed environment, a number of end users may need to access a number of multimedia servers through one or several communication networks. This scenario reveals the requirement for a distributed platform to support user access to multimedia objects according to their QoS requirements while optimizing multimedia server resource utilization. For that purpose SAP has been designed to act as a middleware between media-on-demand clients and servers. SAP architecture is distributed and contains two basic components. The first one, at the client host, acts as the user access interface. It controls user access to the requested multimedia objects and checks if the required QoS parameters are satisfied. The second component, at the server level, interfaces with the multimedia objects' storage server, e.g., a continuous media file server. It manages access requests to the multimedia objects supported by this server and keeps track of the server load information. Hereafter, the first component will be referred to as the user agent and the second one as the server agent. Furthermore, depending on the networked system topology, the platform can contain a number of additional management components acting as intermediates between user agents and server agents. These intermediate managers allow the directing of user requests to the most appropriate servers in a transparent and dynamic manner. Both user QoS requirements and multimedia servers' availability are taken into account in the process of directing users' requests. Note that in SAP, it does not matter what the QoS parameters are exactly. These depend on the features of the multimedia system to be accessed using SAP; for example, in our implementation of a SMoD system, described in Section V, the QoS parameters managed within SAP and used to access a video object are the frame rate, the colors, and the resolution. The user agent allows the user to set these parameters at the access interface.

Two types of managers are defined for the platform, the service access manager (SAM), and the domain manager. The latter is responsible for a domain, grouping a set of multimedia servers according to different criteria, e.g., geographical constraints. SAM is responsible for a set of domains grouped according to organizational policies. It acts as a smart directory service, gathering both functional and management information on the multimedia domains and makes this information available to domain managers and end users. Functional information concerns mainly the multimedia content supported within the multimedia domains. Management information is primarily used for managing user access to multimedia servers by directing user requests to the appropriate domains based on: load information, QoS requirements, and cost constraints. The domain manager monitors the multimedia servers within its domain through the server agents associated with these multimedia servers. It collects state as well as load information and performs the appropriate directing of user requests. It also detects multimedia server failures whenever they occur. As a result, SAM and domain managers support the appropriate exchange of messages between user agents and server agents. In addition, they maintain state information that allows the determination of the most suitable multimedia server in response to a user request.

The introduced agents and managers can be configured (their number, location, and dependencies) to suit a given physical topology or a given multimedia service provision policy. Based on these agents and managers, the overall SAP platform intervenes between users, i.e., client hosts, and existing/future multimedia servers, i.e., server machines, to increase service availability as well as to optimize resource utilization.

Fig. 1 shows an example configuration of SAP, illustrating the distributed feature of SAP. The example configuration consists of a two-level hierarchy where the service access manager is the root, domain managers constitute the first level, and server agents are the leaves. However, the architecture of SAP may contain as many domain levels as necessary, according to the size and feature of the overall system, e.g., the number of servers, their distribution, the network topology, etc. The performance of the overall system is also an important design choice to determine the number of domain levels in the access platform. Indeed, involving a large number of

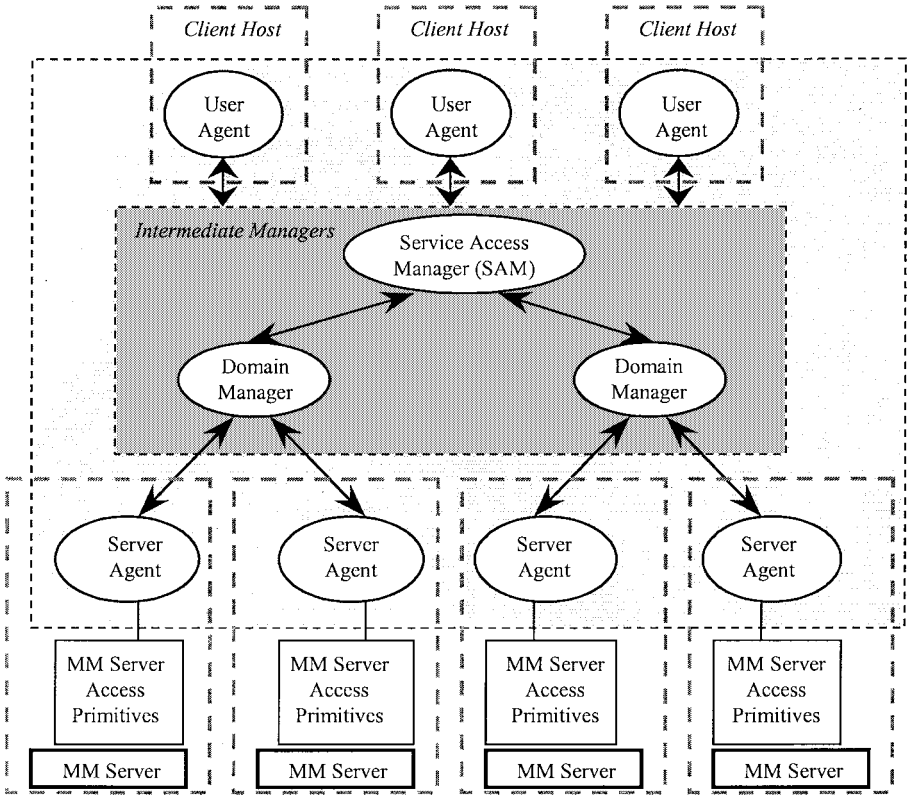

Fig. 1. SAP distributed architecture.

intermediate domain managers in the decision making process will affect the performance in terms the of response time.

Sections III-A–E present in detail the functions provided by SAP components, as well as the messages exchanged between them.

A. The User Agent

The user agent consists of three components: user interface, QoS manager, and a client controller (as shown in Fig. 2). The user interface consists of two main parts. The first one offers a way to search and select a multimedia object from a database. The second one allows the specification of the desired quality for the presentation as well as access cost constraints. It also allows the user to renegotiate the QoS parameters during the multimedia object display.

Whenever a user selects a multimedia object with specific QoS requirements, the user interface component invokes the QoS manager, which starts by checking whether the client machine characteristics, such as the screen size and color, support the requested QoS. If not, the QoS manager sends to the user a reject message, possibly with an alternative offer, through the user interface. In this case, the user might abandon the request, accept the alternative offer if any, or initiate a renegotiation. If the specified QoS parameters conform to the capabilities of the client host, the QoS manager invokes the client controller together with its user requirements, i.e., QoS and the client machine capabilities, like the available decoder. The client controller builds a request message before sending it to the domain manager. It then waits for a response at a specific host port while initiating a timer. If the timer expires, or the user request is rejected by the multimedia-on-demand system, a reject message is displayed at the user interface. Otherwise, the client controller invokes the appropriate multimedia client to display the multimedia object. This is done via a message interpreter responsible for mapping SAP protocol data units to a specific primitive that starts the corresponding multimedia

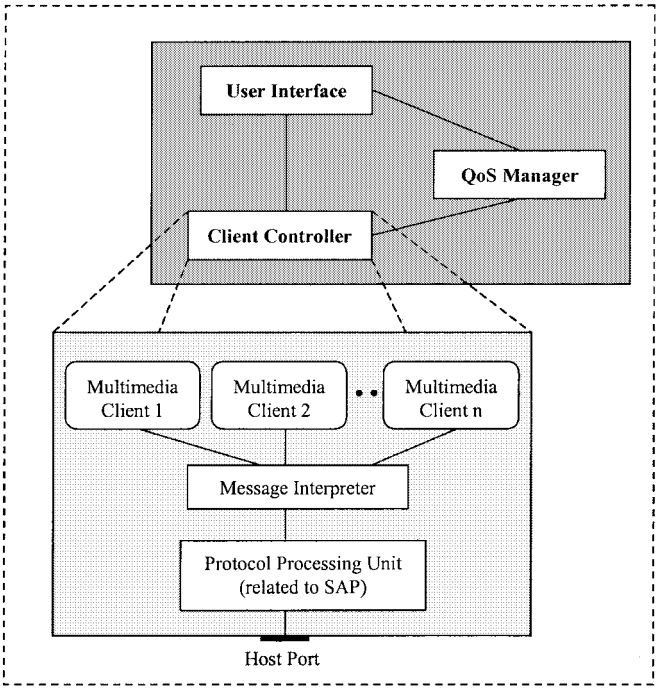

Fig. 2. User agent architecture.

client. The latter can be any of the available commercial or research multimedia displays, like a video or an audio player.

During the multimedia object display, the user may experience a QoS degradation, manifested, for example, as an unacceptable presentation quality. In such an occurrence, the user agent notifies the appropriate domain manager asking for an alternative server capable of delivering the multimedia object with the contracted QoS. In the current implementation, the state of the multimedia object display is registered at the time the display is interrupted, e.g., the position of the last video scene displayed. This allows the alternative server to continue the delivery of a multimedia object started by another server. The new server restarts the multimedia object display based on the state parameters registered earlier.

The object-based design of the user agent allows for higher flexibility and easier maintenance. As an example to support a new multimedia application, one can integrate the appropriate user interface without changing the existing objects of the user agent.

B. The Server Agent

A server agent is delegated to represent a multimedia server within SAP. The server agent continuously waits at a specific service port for requests coming from its domain manager. When a request is received, the server agent checks first the capacity of the multimedia server to deliver the requested multimedia object with the desired QoS. This leads to either starting the delivery by the server or sending a reject message to the user. The server agent is composed of two main parts or functions. The first part deals with the communication with other SAP components like the domain manager and the user agent. The second part is related to the multimedia storage server represented by the server agent in the SAP. Its implementation is specific to the encapsulated multimedia server. The advantage of this design is the ability to change one multimedia server to another by changing only the corresponding part in the server agent. Similarly, supporting a new multimedia server by the SAP necessitates the provision of the corresponding interoperability functions at the server agent.

In the context of the SAP service requests management, a server agent exchanges messages with the domain server manager concerning its operational state and its availability. This allows the SAP to detect faulty multimedia servers. For this purpose, the server agent periodically notifies the domain manager of its availability to handle service requests. Notification frequencies need not be equal for all server agents in the system. The notification frequency depends on a number of factors, such as the reliability of the server; in addition, it takes into account how much importance we place on these factors. Such factors can be computed according to collected statistics on the past behavior of the server.

The load information of a multimedia server, referred to as the server availability, is computed as the number of service requests the server can handle with a given QoS at a given time. This highly depends on the internal implementation of the multimedia server such as the processing architecture, storage capacity, access performance, etc. The server agent computes the load of the multimedia server it encapsulates, based on the load model described in Section IV-A.

C. SAM

The role of the SAM is to redirect user requests to the appropriate multimedia domain—that is, the domain containing the most appropriate multimedia server to handle the request. For this purpose, SAM maintains two state information tables, namely SAM_MMObject_Table and SAM_Load_Table. SAM_MMObject_Table contains two attribute types: MMObject_Id and List_of_Domains. For each multimedia object identified by MMObject_Id, it gives the list of domains containing at least one instance of this object. Each domain in the List_of_Domains is identified by its Domain_Id. SAM_Load_Table gives the load of this domain for each domain, identified by a Domain_Id.

Upon receipt of a request from a domain manager or user, SAM determines the domains with at least one multimedia server storing the requested multimedia object. It uses SAM_MMObject_Table for this purpose. The load information contained in SAM_Load_Table is sorted from the lightly loaded domain to the highly loaded one. According to these tables, domain managers are successively requested to deliver the multimedia object. A reject message is sent back if none of the domains can satisfy the request.

The load information for each domain maintained by SAM is received from the corresponding domain manager. The latter computes the domain load as the weighted average sum of the load levels of the various multimedia servers within the domain.

D. The Domain Manager

Domains are logical structures used as flexible means for clustering multimedia servers in order to control resources utilization. Each domain contains a domain manager, which maintains the global view of the encapsulated multimedia servers. Based on global and timely knowledge of servers state, a domain manager can offer end users better quality and fault tolerant access to the multimedia objects supported by these servers. The state information on a server concerns mainly its operational state, e.g., out of service, and its availability, e.g., load, response time, etc.

As described previously, domains can be defined according to geographical, organizational, and service performance as well as other criteria. The various domains are organized into a hierarchy of domains to provide a system-wide access to multimedia objects. The domain structuring allows to reduce the management complexity of the overall multimedia-on-demand system. Indeed, a domain manager performs its management task autonomously, but may cooperate with other domain managers in the context of the global access management task. The number and size of the domains can be determined based on several factors, such as investment versus revenue for the service provider and/or quality versus price of the offered services to end users.

Similar to SAM, but within a single domain, the domain manager redirects user requests to the appropriate server. It maintains mainly two state information tables,

namely, D_MMObject_Table and D_Load_Table. Emphasis is given on the load distribution capability. Therefore, the state information is limited to servers' load information. D_MMObject_Table contains two attribute types: MMObject_Id and List_of_Server_MMO. It gives, for each multimedia object identified by MMObject_Id, the list of servers storing a copy of the multimedia object together with the QoS characteristics of this copy. Each item in the List_of_Server_MMO is composed of the Server_Id storing a copy of the object and a list of QoS characteristics of the object copy. D_Load_Table gives the load of this server for each server, identified by a Server_Id.

Upon receipt of a request from a user agent or from SAM, the domain manager determines from D_MMObject_Table the servers storing the requested multimedia object with the QoS characteristics required by the user and supported by the user host. The domain manager compiles the load information contained in D_Load_List to determine the most appropriate multimedia server with respect to the user request. For that purpose, D_Load_List is sorted from the lightly loaded server to the highly loaded one. In practice, the domain manager submits the user request to the server agent responsible for the least loaded server containing a copy of the multimedia object. This process is repeated until a server succeeds in delivering the requested multimedia object or all the servers fail. In the latter case, a notification is sent to SAM.

The load information for each multimedia server maintained by the domain manager is received from the corresponding server agent. The latter computes the load of the server it is responsible for, based on the load model described in Section IV-B.

E. Interactions Between SAP Components

The components described in the previous sections interact with each other through a message passing mechanism to provide the overall SAP function. The following signaling messages are defined to support these interactions.

- Service_Inq (Sender_Id, User_Id, MMObject_Id, QoS, Host_Capabilities): this message is sent by the user agent, SAM, a domain manager, or a server agent with the following parameters. Sender_Id identifies the sender, User_Id indicates the IP address of the user's host and the used port number, MMObject_Id uniquely identifies the requested multimedia object, QoS indicates user QoS requirements that can be directly expressed in terms of human-perceptible quantities, and Host_Capabilities indicates the static capabilities of the user's host, such as the monitor size and resolution or the supported compression schemes.
- Service_Res (User_Id, MMObject_Id, Status): this signal is sent by a server agent or a domain manager, in response to Service_Inq(), with the three following parameters. User_Id, MMObject_Id, and Status indicate whether the user request can be supported (Status= ACCEPT) or not (Status= REJECT).
- Service_Conf (User_Id, MMObject_Id, Status, Proposition): this message is initially sent by a server agent to the user with four parameters. User_Id, MMObject_Id, and Status indicate whether the user accepts the Proposition (Status= ACCEPT) or not (Status= REJECT), and Proposition indicates how the server agent can deliver the multimedia object (e.g., QoS characteristics) identified by MMObject_Id.
- Service_Viol (Sender_Id, User_Id, MMObject_Id, QoS, Host_Capabilities): this signal is sent by a server agent when the server cannot maintain the delivery of the multimedia object (identified by MMObject_Id) to the user (identified by User_Id) while satisfying the user QoS requirements QoS. It is also sent by the user, identified by User_Id, when he/she notices a QoS degradation of the presentation of the multimedia object identified by MMObject_Id.
- Service_Alive (Sender_Id): this signaling message is sent by a server agent, identified by Sender_Id, to its domain manager. It is used to detect server failures.
- Add_MMObject (Sender_Id, MMObject_Id, MMObject_Characteristics): this message is sent by a server agent to a domain manager or by a domain manager to SAM when a new multimedia object is created. It contains three parameters. Sender_Id, MMObject_Id, and MMObject_characteristics, which indicates the static characteristics of the multimedia object, such as QoS constraints (e.g., color and resolution) and compression format (e.g., MPEG or MJPEG).
- Delete_MMObject (Sender_Id, MMObject_Id): it is sent by an agent server to a domain manager or by a domain manager to SAM when an existing multimedia object is deleted.
- Update_MMObject (Sender_Id, MMObject_Id, MMObject_Characteristics): this message is sent by a server agent to a domain manager when the characteristics of an existing multimedia object are altered.
- Update_Load (Sender_Id, Load_Level): this signaling message is sent by a server agent, identified by Sender_Id, to a domain manager to report that the current load of the server has reached Load_Level. It is also sent by a domain manager, identified by Sender_Id, to SAM notifying that the current load of the domain has reached Load_Level.

These primitives assume a global naming scheme in the distributed system to uniquely identify the various involved entities including multimedia objects and SAP components. Some exchanges between SAP components using service primitives are shown in Fig. 3. The exchanges shown are only for illustration purposes and do not depict any specific scenario.

IV. QOS SENSITIVE LOAD DISTRIBUTION

A. Load Model for Media Servers

One of the main criteria used to find the most appropriate server to which to redirect users service requests is the load of multimedia servers. Usually, a server's load is computed as the number of processes currently executed divided by the server's speed, or equivalently, the number of service requests

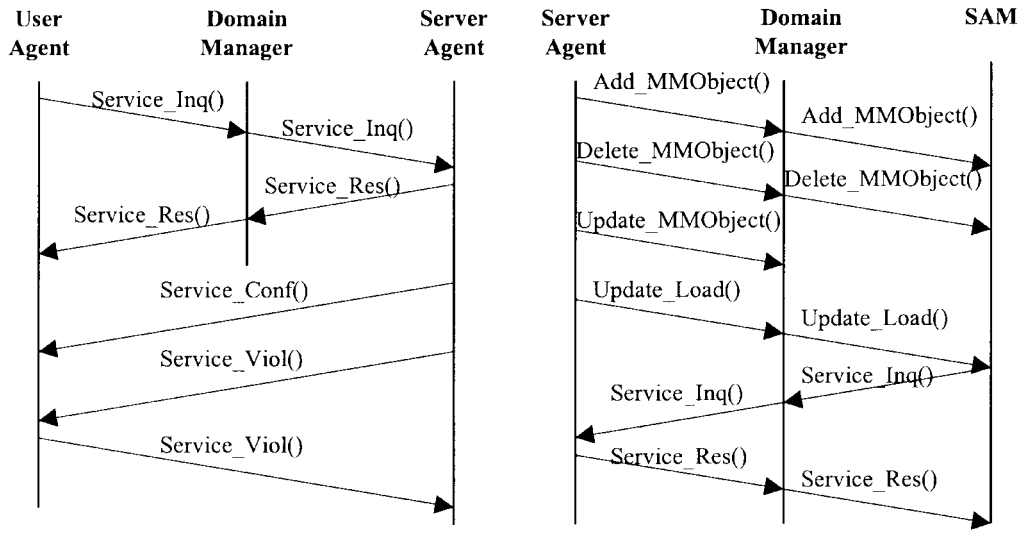

Fig. 3. Interactions between SAP components.

currently handled divided by the server's speed (a function of the CPU and disk access speeds).

In a multiservice environment supporting different QoS, the load information, as defined earlier, does not reflect the real load and may lead to inefficient placement decisions. Indeed, the "least loaded server," handling the smallest number of service requests, may not correspond to the server which has the smallest load volume. For example, a server running a process that requires a large amount of resources, e.g., decoding and displaying a high resolution video, is effectively more loaded than a server running tens of light processes, e.g., text processing. Furthermore, there are no guarantees that the selected least loaded server can support the user QoS requirements.

To alleviate these limitations, we propose a scheme that provides accurate information about the capacity of a server to support a given request by performing extensive measurements prior to service operation. This scheme allows to determine the server which is lightly loaded, and it is able to support the user request with the desired QoS.

In the proposed scheme, QoS requirements are grouped into QoS classes. Each class represents a range of QoS parameters' values. A service request is associated with a QoS class, and thus, it requires a certain amount of resources from the server. A load model with similar properties than the one described here can be found in [24]. However, the model in [24] is introduced to characterize the load of a network link in a multiclass broadband environment.

At a given time, the state of a given server can be defined as the set of service instances currently supported by the server. Based on this classification, we can determine through experiments the set of nonblocking, semiblocking, and blocking states for a given server. A server is in a nonblocking (respectively, blocking) state if it can (not) support new service request(s) without affecting the service instances currently supported. When the server is in a semiblocking state, certain new service requests can be supported while others cannot. This QoS classification and server states determination are formalized as follows.

Let us define CL as the set of QoS classes a server can support and n the cardinality of CL.

Definition 1: St is defined as the set of service instances currently supported by server S at time t. It reflects the state of a given server at a given time.

We note $St = (V(cl^1), V(cl^2), \cdots, V(cl^n))$, where $V(cl^i)$ is the number of currently provided service instances belonging to QoS class $cl^i; 1 <= i <= n$ and $n = \text{Cardinal}(CL)$.

Definition 2: A finite state machine (FSM) that represents a given server S is a tuple (S, S^0, SR, TSR, T), where:

- S is the set of states in which the server can still accept new requests (nonblocking states);
- S^0 is the initial state of the server;
- SR represents the set of service requests submitted to the server;
- TSR is a set of received service termination requests generated by users;
- T: $S\{S^0\} * SR \rightarrow S; S\{S^0\} * TSR \rightarrow S$ is a transition function.

The transition function T operates as follows.

When the server S receives a service request, for example sr, belonging to QoS class cl^k at time $t + \delta$, two options are possible: either the server is in a blocking state and rejects the sr request, or it is in a nonblocking state and processes this request. In the latter case the server transits from state St to $St' = [V(cl^1), \ldots, V(cl^k) + 1, \ldots, V(cl^n)]$.

When the server receives a service termination request at time $t + \delta$, say tsr, corresponding to a service request of QoS class cl^k, it transits from state St to $St'' = [V(cl^1), \ldots, V(cl^k) - 1, \ldots, V(cl^n)]$.

Fig. 4 shows a simple example of the FSM of a server supporting only two classes of QoS cl^1 and cl^2. The server is said to be in state (n, m) when it is currently providing n service instances of class cl^1 and m service instances of class cl^2.

According to the example FSM, states (3,2), (4,1), (2,3), and (1,4) are blocking states (dark grey circles in Fig. 4), which means that all new requests received by the server, while in one of these states, are simply rejected. State (4,0)

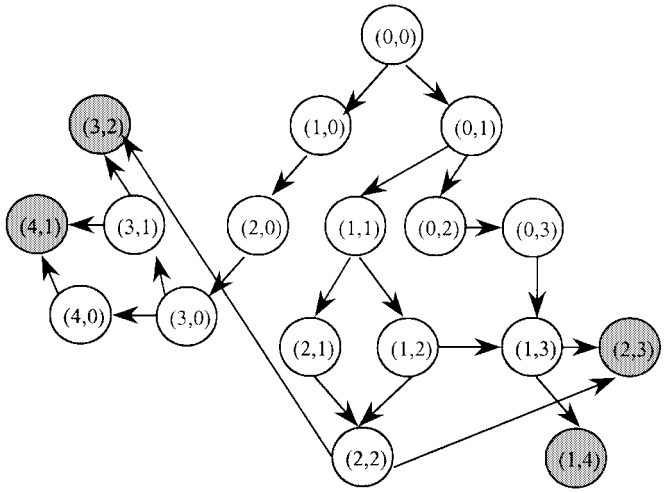

Fig. 4. Example of FSM for a server supporting two QoS classes.

is a semiblocking state; that means that the server will reject all new requests belonging to QoS class cl^1, but can provide only one new service instance of QoS class cl^2. Similarly, states (3,1), (1,3), and (2,2) are semiblocking states represented by grey circles in Fig. 4. The remaining server's states are nonblocking, which means that the server can handle new service requests from both QoS classes. For instance, if the server is in state (1,2) and receives a service request of class cl^1, it accepts this one and transits to a new state, namely (2,2). White circles in Fig. 4 represent nonblocking states.

B. Domain Load

As stated previously, the target scalable access platform introduces the domain concept as a means for grouping multimedia servers to obtain aggregated views of the system resources and hence a more easily manageable distribution of the overall system load. Typically, domains are used to redirect user requests to appropriate multimedia servers based on the global state information maintained at these domains. In addition to the load of each multimedia server in the domain, we may need to compute the load of the domain as a whole. This is particularly needed if a hierarchical domain structuring is adopted for the system where domains can be members of upper level domains. In this case the load of the domains is used to decide the directing of service requests to the most appropriate domain.

The load information of a domain is a function of the load of the multimedia servers contained in this domain. In the next section, a simple approach is used to compute a domain load as the weighted average sum of the load levels of the domain servers. The selection of the weights associated to the various servers depends on several factors, such as the reliability and availability of the servers. These factors can be statistically determined based on the past behaviors of the servers. In practice, the designer of the multimedia-on-demand system assigns weights to the media servers involved in the system. Weights are first statically assigned based on either the designer servers' exploitation policy or the characteristics of the servers or both. The characteristics of the servers are obtained from the server's vendor specifications, such as the performance of the server in terms of processing power, storage capacity, access speed, and so on. The weights are then dynamically adjusted according to statistics on the servers' operation and behavior. The domain manager keeps track of the servers' past behaviors by maintaining in a log several information attributes concerning the operational state, the administrative state, the health, and the availability of each server. Statistics are computed using the information in the log and used to adjust the weights assigned to the servers, e.g., on a day of the week or on a time of the day basis. The weights can also be adjusted to reflect a new policy of the multimedia-on-demand designer. The load information for each multimedia server maintained at the domain is obtained from this server according to the load model described in Section IV-A.

C. Information Exchange Policy

As mentioned in Section II, there are a number of approaches to exchange state information between the hosts involved in the load distribution process. These mainly implement a polling procedure, which may be initiated by a client host or by a server. In the context of our domain-based structuring of the multimedia-on-demand system, load information exchange is based on a server-initiated polling mechanism implemented for each individual domain. Several variants of server-initiated polling can be envisaged for load information exchange within a domain, which appear in the following list.

1) Each server periodically sends notifications about its current load.
2) A server notifies about its current load whenever it changes.
3) The server sends notifications only when significant load levels are reached.

In order to minimize the number of messages exchanged in the system, the last policy is adopted for the exchange of load information. According to the defined load model, a server does not need to send a load update notification each time it transits to a new state except if this one is a blocking or a semiblocking state. Therefore, only the servers capable of supporting user requests with the desired QoS will be considered during task placement decisions.

The exchanged load information, referred to hereafter as Load_Level (LL), is expressed as

$$\mathrm{LL} = \Sigma\{w_i * b_i, \text{where}$$

- b_i is a Boolean that represents the server state with respect to service requests belonging to cl^i. $b_i = 0$ if the server cannot support any new service request of class cl^i; otherwise $b_i = 1$; initially $t_i = 1$ for $i = \{1, \cdots, n\}$.
- w_i indicates the weight associated with cl^i ($w_i < w_j$ means that $cl^i < cl^j$, which in turn means that service requests from cl^j require more server resources than those from cl^i). Hence, the values of w_i, for $1 <= i <= n$, depend on the classification of $cl^i, 1 <= i <= n$, in terms of grade of service.

D. SAP

The most frequently accessed objects are replicated on a large number of servers. In this perspective, a replication scheme such as the one proposed in [17] can be used. The frequency of accesses to multimedia objects is equally assigned to all objects statically at system startup. Access frequency is then incremented each time the object is invoked. The platform easily determines the access frequency for each object as all requests go through the platform. This is done with a reasonable overhead as the platform is designed to manage access to a relatively small number of objects. Operations defined as part of the SAP access protocol (Section III-E), namely ADD_MMObject, DELETE_MMObject, and UPDATE_MMObject, allow to update the access frequency and dynamically change the replication scheme.

Given a user request to access a multimedia object with given QoS requirements, the role of SAP is to locate the appropriate multimedia server to handle this request. A service access policy is defined to guide SAP in the process of dynamically locating the server capable of delivering the requested multimedia object while satisfying user QoS requirements. Several factors are taken into account by the service access policy. These factors are:

1) load of the multimedia servers;
2) QoS characteristics of the multimedia object;
3) user QoS requirements and cost constraints.

The ultimate goal of SAP is to minimize the blocking probability of user requests. Therefore, the SAP service access policy is defined in such a way to avoid rejecting a user request while there are servers in the system that might satisfy this request. The following steps summarize the service access policy implemented by SAP.

1) Identify the servers storing an instance of the requested multimedia object; this gives a list of potential candidate servers.
2) From the list in 1), select the server that has the multimedia object variant satisfying the most user QoS requirements and the server that has the lightest load.
3) Start displaying the multimedia objects; go to step 2) in case the selected server experiences rapid load fluctuation leading to a violation of the service agreement.

A more detailed description of two hierarchical SAP service access policies is given in [25].

V. Implementing a Movie-on-Demand Application Using SAP

Based on the generic SAP, we have developed an application called SMoD to increase the availability of video servers, by providing users with an enhanced access to video movies. SMoD supports a dynamic selection of a video server (VS) that is able to deliver the requested movie with the desired QoS. Upon receipt of the user request to play a movie, SMoD dynamically determines the VS that satisfies the two following conditions:

1) if it contains the requested movie: the movie QoS characteristics should match user QoS requirements and user host display capabilities. For instance, if the user requires TV frame rate to play the movie, and the user host machine supports only MPEG decoding, a server storing the requested movie under MJPEG format or at 15 frames/s is not considered by the domain manager. Metadata on movies, such as resolution and compression format, can be maintained by domain managers in a centralized or distributed repository [25];
2) if it is the "least loaded" among the available VS's.

A user request is rejected, it is only when none of the existing video servers satisfies 1) and 2). SMoD also allows automatic recovery, whenever possible, from QoS degradations during a movie presentation and to mask server failures. The SMoD application is better explained by describing its operation with an example, when a user wants to play a movie. The operation of SMoD involves SAP components, as shown in Fig. 1, and the interactions between them, as shown in Fig. 3.

1) The user selects a movie to play using the user interface provided by the user agent. The user interface allows searching for a movie using key words as well as other search attributes [26]. It also allows users to specify the desired quality for the presentation of the selected movie as well as the cost they are willing to pay.
2) The user agent sends a request to its domain manager.
3) Upon receipt of this request, the domain manager determines the video servers under its authority that store the requested movie with the requested QoS characteristics.
4) Among the VS's determined in 3), the domain manager selects the least loaded VS according to load information collection and exchange policies previously described. The selected VS is then requested to deliver the movie.
5) The VS checks its resource availability and starts the presentation if no major load fluctuations occurred in the meantime. If movie delivery is not possible due to a lack of resources, the domain manager is notified with a reject message.
6) Whenever a request is rejected by a VS, the domain manager asks the next lightly loaded VS to handle the user request. This process is repeated until a VS succeeds to deliver the movie or all the selected VS's fail. In such a case, the domain manager reports to the service access manager about the unsatisfied user request.
7) SAM forwards the user request to the lightly loaded domain in its load table. The domain manager of the selected domain processes the request by performing operations 3)–6). If all the domains managed by SAM are unable to handle the request, SAM sends a reject message to the user.

A. SAP/SMoD Prototype and Deployment Testbed

To evaluate the scalable access platform through the SMoD application, we implemented an experimental prototype [28]. The prototype offers an interface to users to search and select movies, specify their QoS/Cost requirements, and renegotiate the desired QoS during a movie presentation. The current prototype reuses the user interface developed for a news-on-

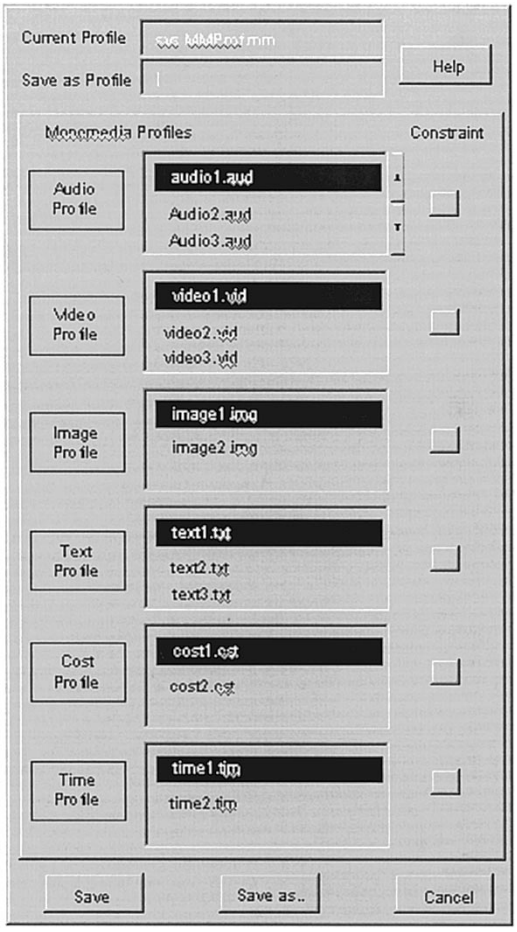

Fig. 5. Component profile window.

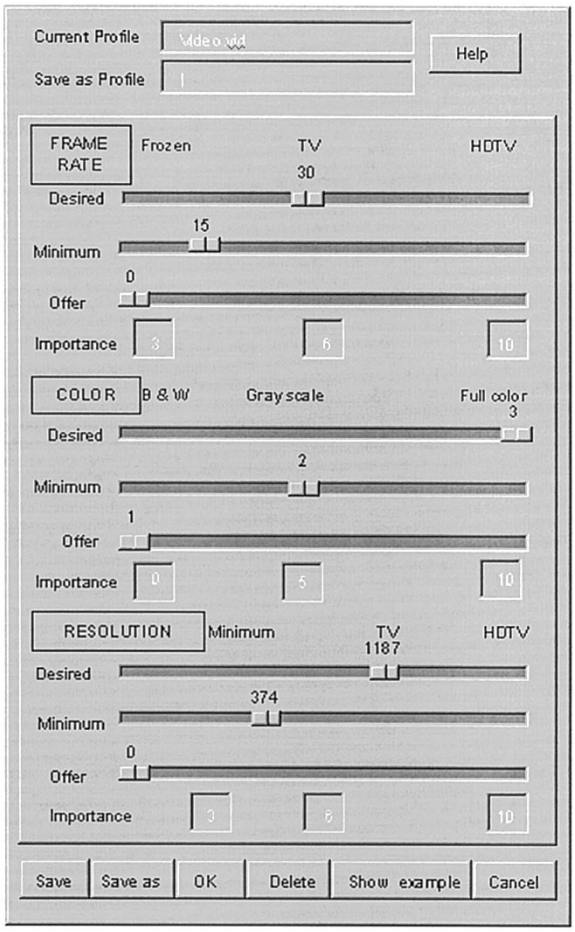

Fig. 6. Video profile window.

demand application [26]. Examples of the interface windows available to the user are shown in Figs. 5 and 6. The profile component window, as shown in Fig. 5, displays the list of monomedia, time, and cost profiles. The user selects the desired profiles by highlighting them. Each profile has an associated customized profile window, which allows the users to specify their requirements. Through scaling bars and predefined values, the user sets the values for QoS parameters, cost, and time, as shown in Fig. 6. For each QoS parameter the user can set the desired value and the minimum acceptable value.

In the current implementation, user agents (UA's), server agents (SA's), domain managers, and the SAM are implemented as Unix processes. These processes, except the UA process, are in an idle state waiting for requests. Unix sockets support communication between processes. When SMoD is launched at a client host, the UA process is initialized and enters an idle state waiting for requests from the user. After the user selects a movie with specific QoS/cost requirements, the UA process checks the validity of the user requirements against the user host characteristics. This is done via a static negotiation procedure that checks the ability of the client machine to support the requested QoS. If the user requirements and user host capabilities do not match, the UA process makes an offer to the user in terms of QoS parameters' values. Otherwise, the UA process assembles the input data into a SAP service access request and sends this request to the domain manager process.

The domain manager process waits for a request by listening at a specific port. Whenever a service request is received at the domain manager port, it is processed and forwarded to the appropriate SA process. The SA process sends the signals specific to the video server in order to reserve the resources necessary for the movie delivery. Only the server resources are considered in this application. Future work is envisaged to use network resources reservation protocols. If enough resources are available to deliver the requested movie, the SA process sends a confirmation message to the UA process, which invokes the appropriate video client, i.e., playback program. Whenever the load level (LL) of the video server, computed by the corresponding SA process, reaches a high threshold, as described in Sections II and III, the SA builds a notification message which is sent to its domain manager for updating load variables. If the selected video server is enable to process the movie delivery as requested by the user, the SA process sends the appropriate signal to the domain manager process. The overall procedure is repeated until a SA process commits to deliver the requested movie, or all the SA processes cannot handle the received user request.

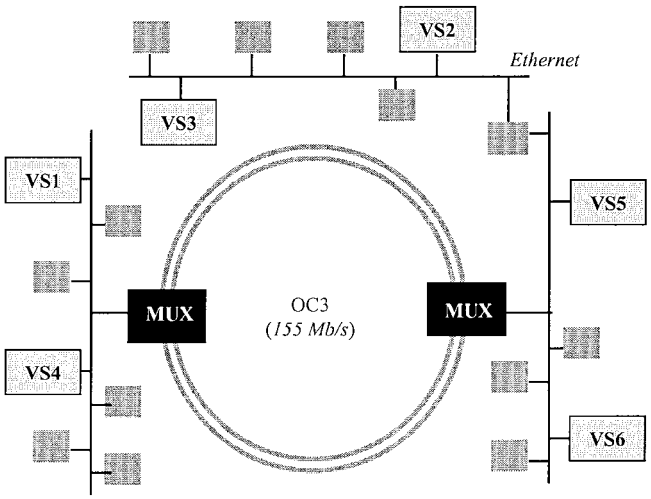

Fig. 7. Experimental testbed.

The prototype has been deployed on an experimental testbed as it is shown in Fig. 7, which includes:

1) an OC3 Sonet ring at 155 Mbits/s;
2) two Sonet OC3 multiplexers with Ethernet cards;
3) three Ethernet segments at 100 Mbits/s;
4) a number of client hosts connected to the Ethernet segments;
5) six video servers distributed in the network as shown in Fig. 7.

Different types of existing VS's are integrated in the testbed, and accessed through the SAP/SMoD demonstrator application. Among them there is a research VS, called continuous media file server (CMFS), designed and implemented at the University of British Columbia [27]. When the SA process associated with CMFS receives a request from its domain manager, it initiates the primitive "Activate_request()" supported by CMFS. This primitive allows checking whether there are enough resources to:

1) retrieve in real-time the data, i.e., the movie satisfying the user QoS requirements, to be transmitted over the network to the user host machine;
2) open the network connection(s) to transmit the movie.

Activate_request() returns a Status to indicate the success or failure of the operation. In the former case, the SA process initiates Play_movie(), and in the latter case, it sends a reject message to the domain manager process. Activate_request() is a proprietary operation defined by CMFS and mapped by SAP in the frame of the SMoD application.

Another example of VS integrated in the demonstrator application, this one a commercial product, is VDOLive On-Demand [28]. This VS is characterized by a fixed maximum number (Maximum_Number) of the concurrent users it can support simultaneously. Therefore, the SA designed to represent this VS within SAP maintains a state variable, Number_of_Users, incremented whenever a play out of a movie is initiated. The SA decrements this variable whenever a movie presentation terminates. When the SA receives a request from its domain manager, it checks if Number_of_Users is equal to Maximum_Number, in which case it sends a reject message to the domain manager.

B. SAP/SMoD Performance Evaluation

Early simulation experiments have been conducted to evaluate the performance of SMoD compared to classical VoD systems. The following parameters are used for the simulation.

- Number of users issuing requests (NU) over a given period of time: a period of 6 h is selected for the experiment.
- User request pattern in time (URP): the 6 h experiment duration is divided into time slots with equal service request probability for all time slots.
- Length of the requested movie (LSM): a process is used to generate length values for the requested movies. LSM is generated randomly within the limits of 60–90 min.
- Movie selection pattern (MSP): a pattern which indicates how users select one of the available movies (e.g., from 50 different movies). Popular movies are usually requested more frequently. The considered default MSP is that 80% of the users select the five most popular movies ($i = 1, \cdots, 5$), 15% of the users select 25 less popular movies ($i = 6, \cdots, 30$), and 5% of the users select the 20 least popular movies ($i = 31, \cdots, 50$). The most popular movies are replicated on all the VS's.
- VS capacity (VSC): in this experiment, only one QoS class is supported by VS's. VSC indicates the maximum number of movies the server can deliver simultaneously.
- User access pattern with respect to different servers (UAP): commonly, the majority of users request the services of well-known servers (e.g., those proven to be reliable and that offer a high-performance service).
- Load level (LL): communicated by a server agent to its domain manager. LL above a certain limit (a threshold) means that the VS is highly loaded and hence cannot process any new service request.

The main metric used for the evaluation of SMoD is the blocking probability computed as the number of rejected service requests divided by the total number of service requests. The server agent rejects a service request due to resource shortage.

The simulations have been conducted in a testbed containing six video servers as in Fig. 7. Two servers, VS1 and VS2, are considered as high-performance servers with a capacity VSC $= 50$, and four servers, VS3, VS4, VS5, and VS6, are considered as regular servers with a capacity VSC $= 20$. During initialization, all servers are idle (LL $= \%$). At any time, a server agent process is listening to the server port. Processes implementing user agents model and simulate the generation of users' requests. They randomly issue service requests by generating values for the parameters introduced previously, i.e., URP, LSM, MSP, and UAP. The sum of the number of requests generated by the various user agent processes is equal to the total number of users considered during the experiments. A large number of user agent processes are launched from different locations, i.e., the testbed machines playing the role of client hosts.

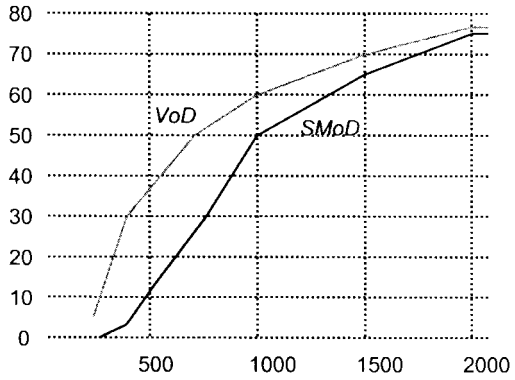

Fig. 8. Blocking probability V's—number of user requests.

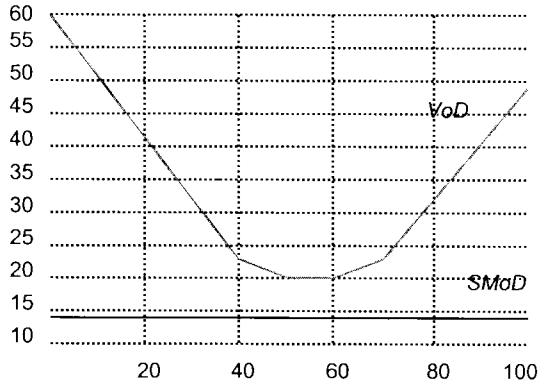

Fig. 9. Blocking probability V's—user's access pattern.

Also implemented as part of the testbed, three processes realize the functions of domain managers. A domain manager process is implemented for each of the three Ethernet segments in the testbed. Therefore, domains have been created in this demonstrator application according to the location criterion. Finally, a process launched on one of the Ethernet segments realizes the service access manager, i.e., SAM.

A number of measurements have been performed to evaluate the blocking probability of service requests when using SMoD and when using a classical VoD system. In a classical VoD system, the user sends a request to a particular server by explicitly supplying the address of this server. If the targeted server is available, the user request is accepted; otherwise, it is rejected. The measurements have been done while alternatively varying the number of user requests, user access patterns, and the proportion of requests received by high-performance (respectively, low-performance) VS's.

Fig. 8 depicts the percentage of requests rejected by SMoD and a classical VoD system as a function of the number of received requests during the 6 h experiment period. The curves show that the number of rejected service requests using a classical VoD increases more rapidly. This is a normal result since in classical VoD systems most of the service requests will hit the high performance server privileged by the users. The high performance server becomes overloaded more quickly and hence starts rejecting requests. On the contrary, in SMoD, users' requests are distributed by the platform transparently to the users, which avoids overloading the high performance server. In this experiment, 80% of user requests are sent to the high-performance VS's and 20% are submitted to the remaining VS's.

The curves in Fig. 9 indicate the percentage of requests rejected by SMoD and VoD, respectively, depending on the distribution of service requests across the various VS's. The percentage of requests received by high-performance and low performance VS's varies from zero to 100% during the 6 h experiment period. These measures show that the variation of user's access patterns, in terms of requests submitted to different servers, has an important impact when using a classical VoD, while it has no impact when using SMoD. This strengthens the advantage of using SMoD as it makes no assumptions on user's access pattern. Classical VoD in contrast tries to statistically or accurately estimate user patterns, which continuously change. Note that the results obtained in the experiments illustrated by Fig. 9 are straightforward. In classical VoD, the users supply the address of the VS, while in the case of SMoD they do not. Hence, the result is naturally better in the case of SMoD. In other words, if SMoD gives the user the possibility to select the VS, the blocking probability will be the same as in a classical VoD system.

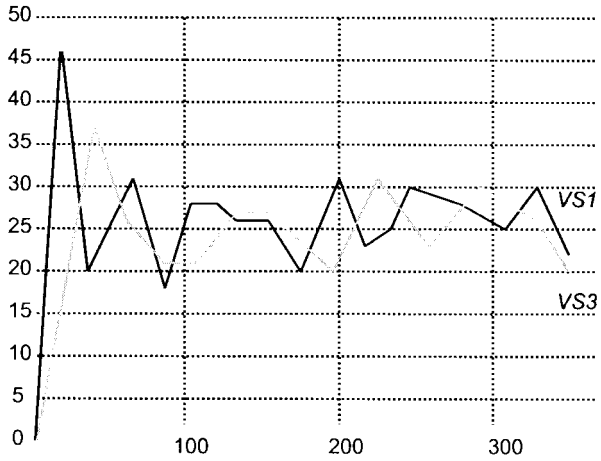

Fig. 10. Proportion of requests supported by VS1 and VS3 using SMoD during the last 20 min of the experiment.

Figs. 10 and 11 show the distribution of requests served by VS1 and VS3 using SMoD and classical VoD, respectively. The graphs indicate the percentage of requests supported by VS1 and VS3 in the last 20 min of the experiment. Two assumptions are made for this experiment: first, the 1000 service requests are distributed along the duration of the experiment; and second, the distribution of service requests submitted to the high-performance VS's and the remaining VS's is 80–20%. The curves demonstrate clearly that the load is better balanced with SMoD as shown in Fig. 10, compared to classical VoD systems as shown in Fig. 11.

In conclusion, the conducted experiments confirmed that more users and service requests can be satisfied using the SMoD application, which in turn relies on the SAP platform to improve the availability and utilization of video servers. The early simulation results consolidate the idea of building a service access platform, which can accommodate a large number of users for a reasonable cost. That is to provide a means to integrate existing and future multimedia server's technology while distributing dynamically and transparently the load across the servers. It is obvious that there is a price

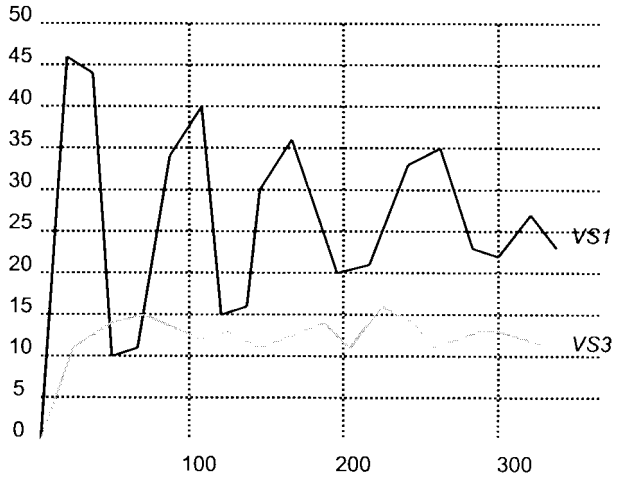

Fig. 11. Proportion of requests supported by VS1 and VS3 using classical VoD during the last 20 min of the experiment.

to pay using applications like SMoD. Due to the overhead introduced by SAP processes, the response time of SMoD is longer than the one of a classical VoD system. However, we believe that the user can tolerate the small response time extension introduced by SAP operation. Indeed, for most applications, the average user will not notice this response time extension. For example, the response time in SMoD remains negligible compared to the average service duration (a movie display is between 60–90 min).

VI. Conclusion and Directions for Future Research

A large number of studies are conducted on enabling technologies for multimedia-on-demand systems. However, these are mainly focusing on the performance of multimedia servers. This can be typified either by emphasis on full hardware implementations and utilization of super computer and massively parallel memory and I/O systems, or more recently, by the design of sophisticated distributed multimedia storage systems supporting intelligent placement schemes for video objects. Yet scalability of multimedia-on-demand systems is still an open issue due to the ever-increasing interest and user demand for multimedia applications.

This paper has presented a system approach to provide scalable multimedia-on-demand systems capable of integrating existing and future multimedia servers. It introduced a generic distributed platform for the management of user access to geographically dispersed multimedia servers. The basic idea behind the design of the SAP is to introduce intermediate access agents and managers between multimedia clients and servers. The intermediate agents and managers are judiciously distributed in the system so as to redirect user requests to the most appropriate servers. The replication of different variants of multimedia objects and the load distribution capabilities of SAP allow it to handle more service requests and hence to support more users while satisfying their QoS requirements and cost constraints.

The platform is generic and flexible. It is independent of any multimedia server technology and can support different types of multimedia applications, such as VoD, news on demand, and others. It can evolve easily to integrate new load distribution policies, to support application-specific replication schemes or to integrate new server technology. For example, integrating a new multimedia server will simply consist of providing, in the server agent, the software module which maps SAP access protocol data units to the primitives of the encapsulated multimedia server.

This paper has also shown an example application of the platform to provide a SMoD. Early performance measures have shown that SMoD is more efficient than classical VoD systems because it reduces the blocking probability of user requests. This allows for a higher user satisfaction and larger revenue for the service provider. The measures have also shown that the nonuniform distribution of user requests, which is the major obstacle for scalability of VoD systems, has almost no impact on SMoD. SAP overcomes the scalability problem by providing a simple and easy-to-implement SAP. The overhead introduced by SAP operation remains negligible compared to the obtained gain. It affects the response time in a way that is not noticeable by average users for most applications.

For future work, the following extensions of SAP are considered.

- Consider network resources and transport capacity as part of the load balancing policy.
- SAP access protocol is hierarchical, i.e., at each level the control is centralized which may lead to bottlenecks. One option will be to add a logical ring structure and the corresponding load distribution protocol at the server agent and/or the domain manager levels of the hierarchy.
- It is more likely that every set of multimedia servers made available to end-users belongs to a given service provider. Mutual cooperation agreements can be made between different service providers. This will necessitate a federation of distinct SAP platforms realized at the level of the service access managers.
- Despite the scalable access scheme provided by SAP, a user request with specific QoS/Cost requirements can be rejected by the system due to resources shortage. One option is to provide a negotiation mechanism for immediate delivery with a lower grade of service or reservation of resources for future delivery.
- The implemented prototype is being enhanced to integrate a Web browser as user interface and to extend CORBA trading facility to implement SAP domain managers and service access manager.

Acknowledgment

The authors wish to thank the reviewers and the editors for their valuable comments that contributed to the improvement of this paper.

References

[1] D. Pegler, D. Hutchison, P. Lougher, A. Scott, and D. Shepherd, "Scalability issues for a networked multimedia storage architecture," *Multimedia Tools and Applications*. [Online]. Available WWW: http://www.comp.lancs.ac.uk/computing/users/phillip/scams.html.

[2] D. Eager and E. Lazowska, "Adaptative load sharing in homogeneous distributed systems," *IEEE Trans. Software Eng.*, vol. 12, no. 5, pp. 662–675, 1986.
[3] N. Shivaratri, P. Krueger, and M. Singhal, "Load distributing for locally distributed systems," *IEEE Trans. Comput.*, vol. 25, pp. 33–44, Dec. 1992.
[4] T. Wang and R. Morris, "Load sharing in distributed systems," *IEEE Trans. Comput.*, vol. 34, pp. 204–217, Mar. 1985.
[5] T. Casavant and G. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Trans. Software Eng.*, vol. 14, no. 2, pp. 141–154, 1988.
[6] O. Kremien and J. Kramer, "Methodical analysis of adaptive load sharing algorithms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 3, pp. 747–760, Nov. 1992.
[7] D. Eager and E. Lazowska, "A comparison of receiver-initiated and sender-initiated adaptive load sharing," *Performance Evaluation*, vol. 6, pp. 53–68, 1986.
[8] R. Boutaba, "A methodology for structuring management of networked systems," in *Proc. IFIP Trans.*, 1994, pp. 225–242.
[9] R. Mirchandaney and D. Towsley, "Adaptive load sharing in heterogeneous systems," *J. Parallel Distrib. Comput.*, vol. 9, pp. 331–346, Aug. 1990.
[10] R. Boutaba and B. Folliot, "Load balancing in local area networks," in *Proc. IFIP Trans.*, 1993, pp. 67–78.
[11] T. Chiueh and R. Katz, "Multi-resolution video representation for parallel disk arrays," *ACM Multimedia*, pp. 401–409, 1993.
[12] M. Kim, "Synchronised disk interleaving," *IEEE Trans. Comput.*, vol. C-35, pp. 978–988, Nov. 1986.
[13] A. Reddy and P. Banerjee, "An evaluation of multiple-disk I/O systems," *IEEE Trans. Comput.*, vol. 38, pp. 1680–1690, Dec. 1989.
[14] P. Lougher, D. Pegler, and D. Shepherd, "Scalable storage servers for digital audio and video," in *Proc. Inst. Elec. Eng. Int. Conf. Storage and Recording Systems 1994*, University of Keele, pp. 5–7.
[15] B. Liskov, S. Ghemawat, R. Gruber, P. Johnson, L. Shrira, and M. Williams, "Replication in the Harp file system," *ACM Symp. Operating Systems Principles*, Pacific Grove, 1991.
[16] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere, "CODA: A highly available file system for a distributed workstation environment," *IEEE Trans. Comput.*, vol. 39, pp. 447–459, 1990.
[17] A. Dan and D. Sitaram, "An online video placement policy based on bandwidth to space ratio (BSR)," in *Proc. ACM SIGMOD'95*, San Jose, CA, pp. 376–385.
[18] D. Brubeck and L. Rowe, "Hierarchical storage management in a distributed video-on-demand system," *IEEE Multimedia*, vol. 3, pp. 37–47, 1996.
[19] J. W. Bolosky, R. P. Fitzgerald, and J. R. Douceur, "Distributed schedule management in the Tiger Video Fileserver," *SOSP*, pp. 212–223, 1997.
[20] D. Patterson, G. Gibson, and R. Katz, "A case for redundant arrays of inexpensive disks (RAID)," *ACM SIGMOD*, pp. 109–116, 1988.
[21] A. Larsen, J. Olkin, and M. Porter, "Oracle media server: Providing consumer based interactive access to multimedia data," *ACM SIGMOD*, pp. 470–477, 1994.
[22] M. Nelson, M. Linton, and S. Owicki, "A highly available, scalable ITV system," *SOSP 15*, pp. 54–67, 1995.
[23] C. Bisdikian, and B. Patel, "Cost-based program allocation for distributed multimedia-on-demand systems," *IEEE Multimedia*, vol. 3, no. 3, pp. 62–72, 1996.
[24] J. M. Hyman, A. A. Lazar, and G. Pacifici, "A separation principle between scheduling and admission control for broadband switching," *IEEE J. Select. Areas Commun.*, vol. 11, pp. 605–616, May 1993.
[25] A. Hafid and R. Boutaba, "A scalable access scheme to multimedia documents with QoS guarantees," in *Proc. IDMS'97*, Darmstadt, Germany, pp. 120–127.
[26] J. Wong, K. Lyons, R. Velthuys, G. Bochmann, E. Dubois, N. Georganas, G. Neufeld, T. Ozsu, J. B. Skelle, D. Evans, A. Hafid, N. Hutchinson, P. Inglinski, B. Kerherve, L. Lamont, D. Makaroff, and D. Szafron, "Enabling technology for distributed multimedia applications," *IBM Syst. J.*, vol. 36, no. 4, pp. 489–507, 1997.
[27] G. Neufeld, D. Makaroff, and N. Hutchison, "Design of a variable bit rate continuous media file server for an ATM network," in *Proc. IS&T/SPIE'96*, San Jose, CA, pp. 370–380.
[28] A. Hafid, P. Dini, M. Hafid, and R. Boutaba, "A scalable video-on-demand system: Architecture and implementation," in *Proc. ICCC'97*, Cannes, France, pp. 262–281.
[29] VDOLive On-Demand. [Online]. Available WWW: http://www.vdo.net/corporate/awards.html.

Raouf Boutaba received the Engineer Diplomat in computer engineering from the University of Annaba, Algeria, in 1988, and the M.S. and Ph.D. degrees from the University Pierre & Marie Curie–Paris 6, Paris, France, in 1990 and 1994, respectively.

Until 1995, he was Assistant Professor at Evry University, France and was involved, as a member of lip6 and PriSM Research Laboratories, in the ESPRIT and ACTS projects funded by the European Community. Between 1995 and 1997, he built and led the telecommunication and distributed systems unit at the Computer Sciences Research Institute of Montreal, Canada. He has been an Adjunct Professor at the University of Montreal since 1996 and is currently a Visiting Professor in the Communications Group of the Electrical and Computer Engineering Department at the University of Toronto, Ont., Canada. His research interests include integrated network and systems management and its application in programmable/active networks. He has directed several research projects sponsored by the computer and telecommunication industry in these areas. He has served as a Guest Editor of international journals.

Dr. Boutaba started and chaired the first IEEE/IFIP Conference on the Management of Multimedia Networks and Services (MMNS'97) and cochaired the Fourth IEEE Workshop on Features Interaction in Telecommunications (FIW'97), as well as MMNS'98.

Abdelhakim Hafid (M'98) received the M.S. and Ph.D. degrees in computer science from the University of Montreal, Montreal, Canada, on quality-of-service management for distributed multimedia applications, in 1993 and 1996, respectively.

He is an Assistant Professor in the Electrical and Computer Engineering/Computer Science Departments (a joint appointment), University of Western Ontario, Ont., Canada and a Research Director of the Advanced Communication Engineering Centre (a venture established by UWO, Bay Networks, Bell Canada). He is also an Adjunct Professor in the Department of Computer Science, University of Montreal. From 1996 to 1997, he was a Research Staff Member at the Computer Research Institute of Montreal (CRIM) in the Telecommunications and Distributed Systems Division, working in the area of distributed multimedia applications. From 1993 to 1994, he was a visiting Scientist at GMD-FOKUS, Systems Engineering and Methods Group, Berlin, Germany, working in the area of high-speed protocols testing. His current research interests are in Internet and multimedia networking.